

Vitesses d'exécution dans R : comparaisons de base R, dplyr et data.table

Antoine Sireyjol

14 février 2019

Plan de la présentation

- ① Présentation de dplyr et data.table
 - 1.1. Dplyr
 - 1.2. Data.table
 - 1.3. Comparaisons avec base R

Plan de la présentation

- ① Présentation de dplyr et data.table
 - 1.1. Dplyr
 - 1.2. Data.table
 - 1.3. Comparaisons avec base R
- ② Comparaison des vitesses d'exécution
 - 2.1. Étude de cas avec le package nycflights13
 - 2.2. Vitesses d'instruction en fonction de la taille de l'échantillon

Plan de la présentation

- ❶ Présentation de `dplyr` et `data.table`
 - 1.1. `Dplyr`
 - 1.2. `Data.table`
 - 1.3. Comparaisons avec base R
- ❷ Comparaison des vitesses d'exécution
 - 2.1. Étude de cas avec le package `nycflights13`
 - 2.2. Vitesses d'instruction en fonction de la taille de l'échantillon
- ❸ Astuces d'optimisation d'un script R
 - 3.1. Utilisation de `*apply`
 - 3.2. Éviter `ifelse`
 - 3.3. Définition d'une variable à l'intérieur de `summarise`
 - 3.4. `group_by` de `dplyr`

Plan de la présentation

- ❶ Présentation de `dplyr` et `data.table`
 - 1.1. `Dplyr`
 - 1.2. `Data.table`
 - 1.3. Comparaisons avec base R
- ❷ Comparaison des vitesses d'exécution
 - 2.1. Étude de cas avec le package `nycflights13`
 - 2.2. Vitesses d'instruction en fonction de la taille de l'échantillon
- ❸ Astuces d'optimisation d'un script R
 - 3.1. Utilisation de `*apply`
 - 3.2. Éviter `ifelse`
 - 3.3. Définition d'une variable à l'intérieur de `summarise`
 - 3.4. `group_by` de `dplyr`
- ❹ Conclusions sur les comparaisons

Plan de la présentation

- ❶ Présentation de `dplyr` et `data.table`
 - 1.1. `Dplyr`
 - 1.2. `Data.table`
 - 1.3. Comparaisons avec base R
- ❷ Comparaison des vitesses d'exécution
 - 2.1. Étude de cas avec le package `nycflights13`
 - 2.2. Vitesses d'instruction en fonction de la taille de l'échantillon
- ❸ Astuces d'optimisation d'un script R
 - 3.1. Utilisation de `*apply`
 - 3.2. Éviter `ifelse`
 - 3.3. Définition d'une variable à l'intérieur de `summarise`
 - 3.4. `group_by` de `dplyr`
- ❹ Conclusions sur les comparaisons
- ❺ Références

Dplyr et data.table

Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R

Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble

Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble
- Syntaxe caractéristique et concurrente des fonctions de base R avec dplyr

Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble
- Syntaxe caractéristique et concurrente des fonctions de base R avec dplyr
- Chaînage possible des instructions avec %>%

Dplyr et le tidyverse

- Tidyverse : environnement d'analyse de données en R
- Propre format de données : le tibble
- Syntaxe caractéristique et concurrente des fonctions de base R avec dplyr
- Chaînage possible des instructions avec %>%
- Très lisible et optimisé

Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables

Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.

Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.

Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.
- `select(data, var1 : varX)` sélectionne certaines variables dans une base.

Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.
- `select(data, var1 : varX)` sélectionne certaines variables dans une base.
- `group_by(data, var)` regroupe une table par une variable

Syntaxe dplyr (1)

La grammaire dplyr s'appuie sur des fonctions aux noms explicites :

- `mutate(data, newvar1 = fonction(var1, var2...))` et `transmute(data, newvar1 = fonction(var1, var2...))` créent de nouvelles variables
- `filter(data, condition)` sélectionne au sein d'une table certaines observations.
- `arrange(data, var1, descending var2,...)` trie une base selon une ou plusieurs variables.
- `select(data, var1 : varX)` sélectionne certaines variables dans une base.
- `group_by(data, var)` regroupe une table par une variable
- `summarise(data, newvar1 = mean(var1), newvar2 = sum(var2))` réalise toute sorte d'opérations statistiques sur une table.

Syntaxe dplyr (2)

- Possibilité de chaîner ces opérations : l'opérateur `%>%`

Syntaxe dplyr (2)

- Possibilité de chaîner ces opérations : l'opérateur `%>%`
- `fonction(data, params...)` est équivalent à `data %>%
fonction(params...)`

Syntaxe dplyr (2)

- Possibilité de chaîner ces opérations : l'opérateur `%>%`
- `fonction(data, params...)` est équivalent à `data %>%
fonction(params...)`
- Exemple :

Syntaxe dplyr (2)

```
library(tidyverse)
# on crée un data frame avec 100 lignes,
# chaque individu appartenant à un des 50 groupes
df <- data.frame(id1 = c(1:100),
                  idgpe = sample(50))

# on y applique les instructions de dplyr
df %>% as_tibble() %>%
  mutate(var = rnorm(100)) %>%
  group_by(idgpe) %>%
  summarise(var_mean = mean(var)) -> output_tibble
print(head(output_tibble), 5)
```

Syntaxe dplyr (2)

```
## # A tibble: 5 x 2
##   idgpe var_mean
##   <int>   <dbl>
## 1     1    0.193
## 2     2   -0.510
## 3     3   -0.345
## 4     4    0.502
## 5     5    0.0766
```

Data.table

- Format optimisé de data.frame

Data.table

- Format optimisé de data.frame
- Complémentaire à base R

Data.table

- Format optimisé de data.frame
- Complémentaire à base R
- Optimisation de l'opérateur [

Data.table

- Format optimisé de data.frame
- Complémentaire à base R
- Optimisation de l'opérateur [
- Chaînage possible des instructions

Data.table

- Format optimisé de data.frame
- Complémentaire à base R
- Optimisation de l'opérateur [
- Chaînage possible des instructions
- Plus lisible, plus rapide que base R

Syntaxe data.table (1)

- l'opérateur [appliqué au data.table change de signification et devient :

`DT[i, j, by]`

Syntaxe data.table (1)

- l'opérateur [appliqué au data.table change de signification et devient :

`DT[i, j, by]`

- `i` permet de sélectionner des lignes de DT

Syntaxe data.table (1)

- l'opérateur [appliqué au data.table change de signification et devient :

`DT[i, j, by]`

- i permet de sélectionner des lignes de DT
- j permet de créer des variables ou d'en sélectionner

Syntaxe data.table (1)

- l'opérateur [appliqué au data.table change de signification et devient :

`DT[i, j, by]`

- `i` permet de sélectionner des lignes de DT
- `j` permet de créer des variables ou d'en sélectionner
- `by` permet de regrouper les traitement selon les modalités d'une variable définie

Syntaxe data.table (1)

- l'opérateur [appliqué au data.table change de signification et devient :

`DT[i, j, by]`

- i permet de sélectionner des lignes de DT
- j permet de créer des variables ou d'en sélectionner
- by permet de regrouper les traitement selon les modalités d'une variable définie
- L'usage de [permet de chaîner les opérations :

Syntaxe data.table (2)

```
library(data.table)
# on convertit notre data frame
# précédemment créé en data.table
dt <- as.data.table(df)

# on y applique les même instructions
dt[, var := rnorm(100)
    ][, list(var_mean = mean(var)),
    by = idgpe] -> output_dt

print(head(output_dt, 5))
```

Syntaxe data.table (2)

```
##      idgpe      var_mean
## 1:      35 -0.54309850
## 2:      13  0.00541226
## 3:      38  0.91597653
## 4:      28 -0.23246660
## 5:       4 -0.28051973
```

Comparaisons avec base R

`dplyr` et `data.table` présentent un certain nombre d'avantages par rapport à l'usage de base R exclusivement :

Comparaisons avec base R

dplyr et data.table présentent un certain nombre d'avantages par rapport à l'usage de base R exclusivement :

- lisibles et - verbeux, grâce notamment au chaînage

Comparaisons avec base R

dplyr et data.table présentent un certain nombre d'avantages par rapport à l'usage de base R exclusivement :

- lisibles et - verbeux, grâce notamment au chaînage
- Pensés pour l'analyse de données

Comparaisons avec base R

dplyr et data.table présentent un certain nombre d'avantages par rapport à l'usage de base R exclusivement :

- lisibles et - verbeux, grâce notamment au chaînage
- Pensés pour l'analyse de données
- Instruction optimisées et bien plus rapides que base R

Comparaison des vitesses d'exécution

Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée

Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables

Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport

Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport
- 26115 lignes et 15 variables

Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport
- 26115 lignes et 15 variables
- On crée `flights_dt` et `weather_dt` avec `as.data.table`

Étude de cas avec nycflights13

- Base `flights` : heures de départ et d'arrivée selon les aéroports + retards au départ et à l'arrivée
- 336776 lignes et 19 variables
- Base `weather` : indications météo, heure par heure, dans chaque aéroport
- 26115 lignes et 15 variables
- On crée `flights_dt` et `weather_dt` avec `as.data.table`
- Étude de cas : fusion des deux tables pour expliquer retards à l'arrivée et au départ en fonction de la météo

Étude de cas avec nycflights13 - Base R

```
flights_time_hour <- aggregate.data.frame(  
  list(arr_delay = flights$arr_delay,  
        dep_delay = flights$dep_delay),  
  list(time_hour = flights$time_hour,  
        origin = flights$origin),  
  mean)  
merge_base <- merge(weather, flights_time_hour,  
                     by = c("time_hour", "origin"),  
                     sort = FALSE)
```

Étude de cas avec nycflights13 - dplyr

```
flights %>% group_by(time_hour, origin) %>%  
  summarise(arr_delay = mean(arr_delay),  
            dep_delay = mean(dep_delay)) %>%  
  inner_join(weather, by = c("time_hour", "origin"))  
  ) -> merge_dplyr
```


Étude de cas avec nycflights13 - dplyr

```
merge_DT <- merge(  
  flights_dt[, list(arr_delay = mean(arr_delay),  
                    dep_delay = mean(dep_delay)),  
    by = list(time_hour, origin)],  
  weather_dt,  
  by = c("time_hour", "origin"))
```

Comparaisons des vitesses de ces instructions

Le package `microbenchmark` nous permet de comparer la vitesse de ces instructions :

```
## Unit: milliseconds
##      expr      min       lq      mean     median
##   base R 650.63653 654.76154 676.31689 667.58741
##   dplyr  50.62595  50.80799  52.26230  51.02246
## data.table 25.18241 25.44809  28.31813  25.68503
##      uq      max neval
## 687.10342 728.84105    10
##  51.69774  62.59707    10
##  26.72809  41.48732    10
```